Noveno Concurso Local de Programación ACM ICPC

Universidad Autónoma Metropolitana Unidad Azcapotzalco

Examen final, 22 a 26 de junio de 2012

Inicio: 10:00 del 22 del junio. Fin: 23:00 del 26 de junio.

Instrucciones: A continuación encontrarás los enunciados de tres problemas. Cada problema tiene un valor de 100 puntos, los cuales se obtendrán de las salidas que tu programa entregue para cada una de 10 entradas distintas. Todos los programas se evaluarán en Linux. El que tu programa funcione con el ejemplo no quiere decir que funcionará siempre. Ningún programa se ejecutará por más de 5 segundos.

Para cada problema que resuelvas deberás enviar por correo el código fuente de un programa a la dirección uam12acm@gmail.com indicando claramente tu nombre completo. De preferencia envía todos tus códigos en un solo correo. El nombre de los archivos que envíes debe ser de la forma xxx.zzz, donde xxx es el nombre del problema y zzz es la extensión, según el lenguaje que hayas usado. Si así lo deseas, puedes resolver cada problema en un lenguaje distinto.

Tu programa deberá leer y escribir exactamente los datos que se indican (ni más ni menos) usando los **archivos** de entrada y salida indicados. En particular, tu programa no deberá: borrar la pantalla, escribir ningún tipo de letrero adicional, esperar ninguna tecla, usar la biblioteca conio, etc.

Quienes programen en C deberán usar el gcc, quienes programen en C++ deberán usar el g++ y quienes programen en Java deberán usar el gcj de la siguiente manera:

```
gcc xxx.c -lm -o xxx
g++ xxx.cpp -lm -o xxx
gcj xxx.java –main=xxx -Wall -o xxx
```

Nota: Todos los puntos calculados que resulten fraccionarios se redondearán siempre hacia abajo

Problema 1: Estudiando lejos del enemigo (100 puntos)

Código fuente: ele.c, ele.cpp, ele.java

Archivo de entrada: ele.ent Archivo de salida: ele.sal

Dos estudiantes de tu salón se caen realmente mal por lo que desean sentarse lo más lejos posible uno del otro. Por alguna razón que nadie comprende, el salón de clases es convexo y sólo hay un asiento disponible en cada una de las **N** esquinas del mismo. Ayuda a tus dos compañeros escribiendo un programa que, dadas las coordenadas de los **N** asientos, calcule en qué asientos deben sentarse para estar lo más alejados posible.

Entrada: Un entero **N** seguido de las **N** coordenadas enteras **X**, **Y** de los asientos. Puedes suponer que $3 \leq \mathbf{N} \leq 1\,000\,000$ y que $-10\,000 \leq \mathbf{X}, \mathbf{Y} \leq 10\,000$.

Salida: Las coordenadas $\mathbf{X_1}$, $\mathbf{Y_1}$ y $\mathbf{X_2}$, $\mathbf{Y_2}$ de los dos asientos, un asiento por línea, en los que tus compañeros deben sentarse para estar lo más alejados posibles. Las componentes de las coordenadas deberán ir separadas por un espacio.

Evaluación: En caso de que las coordenadas de los asientos sean válidas, 10**D**/**M** puntos donde **D** es la distancia entre dichos asientos y **M** es la distancia máxima posible.

Ejemplo de entrada	*Ejemplo de salida*
5 1 0 3 0 3 4 4 2 0 2	1 0 3 4

Problema 2: Pulga binaria saltarina (100 puntos)

Código fuente: pbs.c, pbs.cpp, pbs.java

Archivo de entrada: pbs.ent ***Archivo de salida: pbs.sal***

Una pulga está inicialmente parada sobre el 0 de una recta numérica y le gustaría pararse sobre la marca del número **K**. Para que sea más divertido llegar a dicha marca, la pulga se ha propuesto iniciar con un salto que sea una potencia de 2 y que, en caso de tener que seguir saltando, cada nuevo salto sea del doble de largo que el anterior. Independientemente de la longitud del salto, la pulga puede decidir libremente si el salto lo da hacia la izquierda o hacia la derecha. Escribe un programa que calcule los **S** saltos que debe dar la pulga para llegar a la marca en la que desea pararse.

Entrada: Un entero **K**. Puedes suponer que $0 \leq \mathbf{K} \leq 1\,000\,000$.

Salida: Un entero **S** seguido de una línea que contenga **S** enteros separados por un espacio y que representan, en orden, la longitud y dirección de los saltos de la pulga. Un salto a la izquierda se representa con un entero negativo y un salto a la derecha con un entero positivo.

Evaluación: 3 puntos si los saltos son válidos y la pulga llega a la marca deseada. En ese caso, 7**M/S** puntos adicionales donde **M** es el número mínimo de saltos necesarios para llegar a la marca.

Ejemplo de entrada	*Ejemplo de salida*
2	2 -2 4

Problema 3: Encuadrando la votación electoral (100 puntos)

Código fuente: eve.c, eve.cpp, eve.java

Archivo de entrada: eve.ent Archivo de salida: eve.sal

Próximamente se llevarán a cabo elecciones en tu ciudad y el instituto electoral te ha contratado para que los ayudes a proponer las subregiones electorales. Para fines de la elección, tu ciudad se considera una región rectangular con **N** x **M** viviendas y por cada vivienda se tomará un voto que podrá ser por el partido **A**, **B** o **C.** Para decidir al ganador, tu ciudad será dividida en tres subregiones electorales rectangulares con al menos **R** viviendas cada una y el ganador de cada subregión será el partido con más votos en la misma (en caso de empate nadie es considerado el ganador de dicha subregión). Al final, el ganador de la elección completa será el partido que haya ganado más subregiones electorales que los demás (y en caso de empate, nadie gana).

En días recientes se realizó una encuesta en todas las viviendas por lo que sabes por cuál partido votará cada vivienda y como apoyas al partido **A**, deseas proponer las subregiones electorales de modo que ocurra lo que más le favorezca. Legalmente el único resultado que importa es el de la elección completa (sin importar cómo ocurrió) y obviamente lo que más le favorece a tu partido es ganar, pero en caso de no ser posible, lo que más le favorece es que nadie más gane.

Entrada: Tres enteros **N**, **M, R** seguido de **N** x **M** caracteres **A**, **B**, o **C** que representan el voto de las viviendas a los partidos. Puedes suponer que $2 \leq$ **N, M, R** ≤ 1000 y que siempre será posible generar una división en subregiones válida.

Salida: Una matriz de **N** x **M** dígitos 1, 2 o 3 donde cada dígito representa a qué subregión fue asignada la vivienda correspondiente.

Evaluación: 3 puntos si la división en subregiones es válida. En ese caso, 7 puntos adicionales si dadas las situaciones posibles, ocurre lo que más le favorece a tu partido.

Ejemplo de entrada	*Ejemplo de salida*
3 2 2 AB AC CA	12 12 33